```
char queue[MAX][10];
int head, tail, x, y;
```

```
 <0> Initialize Queue
 <1> Enqueue  (Add)
 <2> Dequeue  (Remove)
 <3> List Queue
 <Q> Quit
Which option <0-3,Q> ?       1
```

```
Qhead      =  0
Qtail      =  0
Qsize      =  0
```

```
<< Append items to the end of the queue >>
   => item to enqueue : orange
   => item to enqueue : apple
   => item to enqueue :
```


完成以下程式 1-6

```
void QueueInit(){
        head=tail=0;
}
```

```
int QueueEmpty(){
        return___// (2)
}
```

```
int Qsize(){
        return___// (1)
}
```

```
int QueueFull(){
        return___// (3)
}
```

```
void displayQueueInfo(){
        gotoxy(50,02);
        printf("QHead = %2d", head);
        gotoxy(50,03);
        printf("QTail  = %2d", tail);
        gotoxy(50,04);
        printf("QSize  = %2d", Qsize());
}
```

```
void enqueue(char item[]){
        if(QueueFull())
                printf("Sorry - Queue Full!\n");
        else{
                ___ // (4)
        }
}
```

```
void dequeue(char item[]){
        if(QueueEmpty())
                printf("Sorry - Queue Empty!\n");
        else{
                printf("Item %10s removed\n",___);
                ___ // (5)
        }
}
```



```
void QueueAdd(){
        char item[10];
        printf("<< Add items >>\n");
        y = whereY();

        do{
                gotoxy(1,y);
                printf("   => item to enqueue : ");
                fflush(stdin);      gets(item);

                y = whereY();
                if(strlen(item)>0) enqueue(item);
                displayQueueInfo();
        }while(strlen(item)>0 && !QueueFull());
}
```

```
void listQueue(){
        int i;
        if(QueueEmpty())
                printf("\nSorry - Queue is empty !\n");
        else{
                printf("\n<< Items in queue >>\n");

                for(i= _____)       // (6)
                   printf("<%02d> %10s\n", i, queue[i]);
        }
        system("pause");
}
```



```
void displayMenu(){
        printf(" <0> Initialize Queue        \n");
        printf(" <1> Enqueue  (Add)          \n");
        printf(" <2> Dequeue  (Remove)       \n");
        printf(" <3> List Queue              \n");
        printf(" <Q> Quit                    \n");
}
```

```
main(){
        char option, item[10];
        QueueInit();
        do{
                clrscr();               displayMenu();
                printf("Which option <0-3,Q> ?  ");
                getxy(&x,&y);           displayQueueInfo();
                gotoxy(x,y);            option=getche();
                option=toupper(option);
                printf("\n\n");
                switch(option){
                        …
                }
        }while(option!='Q');
}
```

→ p.8


```
main(){
        QueueInit();
        do{
```

→ p.7

```
                switch(option){
                        case'0':
                        case'I':        QueueInit();        break;
                        case'1':
                        case'A':        QueueAdd();         break;
                        case'2':
                        case'D':        dequeue(item);      break;
                        case'3':
                        case'L':        listQueue();        break;
                }
        }while(option!='Q');
}
```

請改寫為: Circular Queue??

```
#define MAX 10
char stack[MAX][10];
int top, x, y;
```

```
 <0> Initialize stack
 <1> Push  (Add)
 <2> Pop  (Remove)
 <3> List Stack
 <Q> Quit
Which option <0-3,Q> ?       1
```

```
Top = -1
Stack size =  0
```

```
<< Add items to the top of the stack >>
   => item to push : A
   => item to push : B
   => item to push : C
   => item to push :
```

```
Top =  2
Stack size =  3
```


完成以下程式 1-6

```
void StackInit(){
        top=-1;
}
```

```
int StackEmpty(){
        return____   // (1)
}
```

```
int StackFull(){
        return____   // (2)
}
```

```
void push(char item[]){
        if(StackFull())
                printf("Sorry - Stack Full!\n");
        else{
                ____ // (3)
        }
}
```

```
void pop(char item[]){
        if(StackEmpty())
                printf("Sorry - Stack Empty!\n");
        else{
                printf("Item %10s removed.\n",_____);
                _____ // (4)
        }
        system("pause");
}
```

```
int StackSize(){
        return___   // (5)
}
```

```
void DisplayStackInfo(){
        gotoxy(50,02);
        printf("Top = %2d", top);
        gotoxy(50,03);
        printf("Stack size = %2d", StackSize());
}
```



```
void StackAdd(){
        char item[10];
        printf("<< Add items >>\n");
        y = whereY();

        do{
                gotoxy(1,y);
                printf("   => item to push : ");
                fflush(stdin);       gets(item);
                y = whereY();

                if(strlen(item)>0) push(item);
                DisplayStackInfo();
        }while(strlen(item)>0 && !StackFull());
}
```

```
void ListStack(){
        int i;
        if(StackEmpty())
                printf("\nSorry - Stack empty!\n");
        else{
                printf("\n<< Items in Stack >>\n");

                for(i=_____)      // (6)
                   printf("<%02d> %10s\n", i, stack[i]);
        }
        system("pause");
}
```



```
void DisplayMenu(){
        printf(" <0> Initialize stack  \n");
        printf(" <1> Push  (Add)       \n");
        printf(" <2> Pop  (Remove)  \n");
        printf(" <3> List Stack        \n");
        printf(" <Q> Quit               \n");
}
```

```
main(){
        char opt, item[10];
        int valid;
        StackInit();
        do{
                … DisplayMenu();
                … DisplayStackInfo();
                … opt=toupper(getche());
                switch(opt){
                        case'0': StackInit();     break;
                        case'1': StackAdd();      break;
                        case'2': pop(item);       break;
                        case'3': ListStack();     break;
                }
        }while(opt!='Q');
}
```

→ p.8

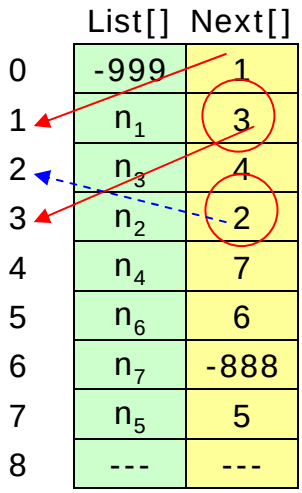

```
do{
        clrscr();               DisplayMenu();
        printf("Which option <0-3,Q> ?  ");
        getxy(&x,&y);           DisplayStackInfo();
        gotoxy(x,y);            opt=toupper(getche());
        valid=(((opt>='0') && (opt<='3'))
                || (opt=='Q'));
        printf("\n\n");

        if(!valid){
                gotoxy(30,24); printf("Error!\n");
                _sleep(1500); continue;
        }
        switch(opt){
        }
}while(opt!='Q');
```

→ p.7

電腦科（1994 年）試卷二 Linked List

(b) 下圖顯示以一表來貯存一正數鏈表 $\{n_1,n_2,n_3,n_4,n_5,n_6,n_7\}$ 的方法。



$\{n_1,n_2,n_3,n_5,n_4,n_6,n_7\}$

(i) 下列各項有什麼目的？ (a)–999 (b)–888

(ii) 就下列各獨立情況，在各空格填上正確數字，以示更新後的鏈表如何在表中貯存。

刪除 n_2 及 n_7 後

	List[]	Next[]
0	-999	1
1	n_1	
2	n_3	
3	n_2	2
4	n_4	
5	n_6	
6	n_7	-888
7	n_5	5
8	---	---

n_4 及 n_5 互換位置後

	List[]	Next[]
0	-999	1
1	n_1	
2	n_3	
3	n_2	
4	n_4	
5	n_6	6
6	n_7	-888
7	n_5	
8	---	---

把 m 加在 n_6 及 n_7 之間後

	List[]	Next[]
0	-999	1
1	n_1	3
2	n_3	4
3	n_2	2
4	n_4	
5	n_6	
6	n_7	-888
7	n_5	
8	m	

只可改變 Next[]的值。

Head = -1, Size = 0

	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	50	70	30	60	20	80	40	90	10	
Next[i]	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Add 50

Head = 0, Size = 1

	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	50									
Next[i]	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Add 70

Head = 0, Size = 2

	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	50	70								
Next[i]	1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Add 30

Head = 2, Size = 3

	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	50	70	30							
Next[i]	1	-1	0	-1	-1	-1	-1	-1	-1	-1

Add 60

Head = 2, Size = 4

	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	50	70	30	60						
Next[i]	3	-1	0	1	-1	-1	-1	-1	-1	-1

Add 20

Head = 4, Size = 5

	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	50	70	30	60	20					
Next[i]	3	-1	0	1	2	-1	-1	-1	-1	-1

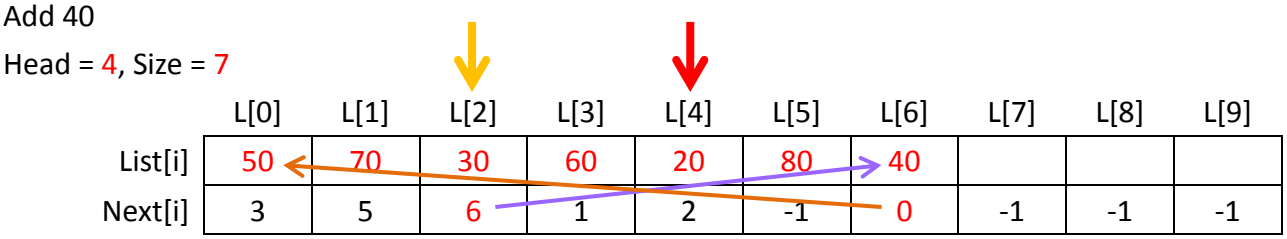
Add 80

Head = 4, Size = 6

	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	50	70	30	60	20	80				
Next[i]	3	5	0	1	2	-1	-1	-1	-1	-1

Add 40

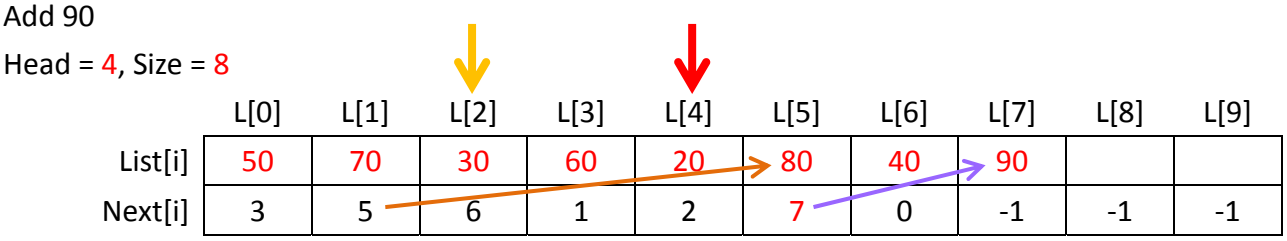
Head = 4, Size = 7



	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	50	70	30	60	20	80	40			
Next[i]	3	5	6	1	2	-1	0	-1	-1	-1

Add 90

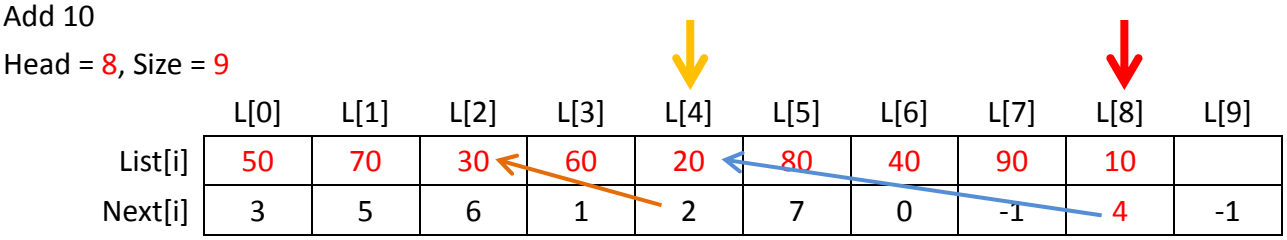
Head = 4, Size = 8



	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	50	70	30	60	20	80	40	90		
Next[i]	3	5	6	1	2	7	0	-1	-1	-1

Add 10

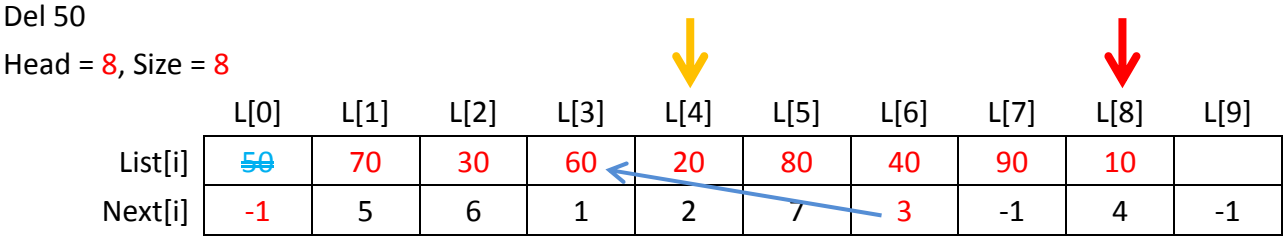
Head = 8, Size = 9



	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	50	70	30	60	20	80	40	90	10	
Next[i]	3	5	6	1	2	7	0	-1	4	-1

Del 50

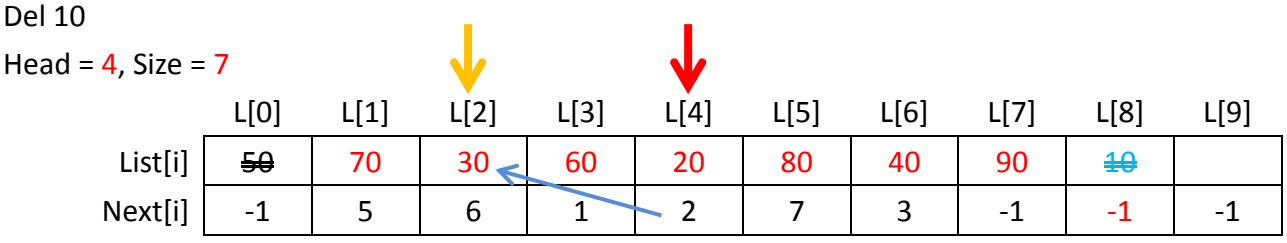
Head = 8, Size = 8



	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	~~50~~	70	30	60	20	80	40	90	10	
Next[i]	-1	5	6	1	2	7	3	-1	4	-1

Del 10

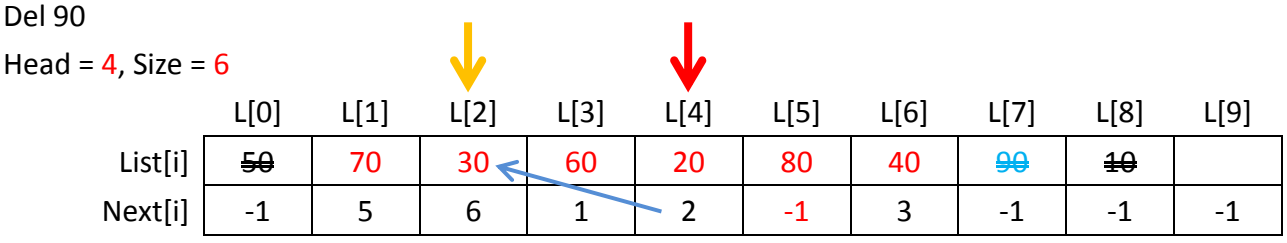
Head = 4, Size = 7



	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	~~50~~	70	30	60	20	80	40	90	~~10~~	
Next[i]	-1	5	6	1	2	7	3	-1	-1	-1

Del 90

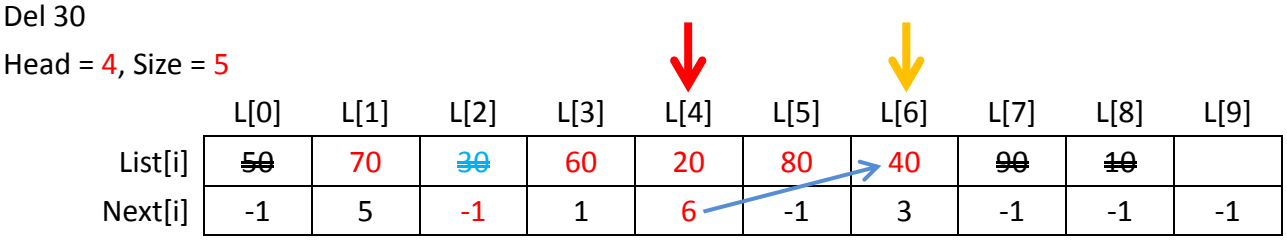
Head = 4, Size = 6



	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	~~50~~	70	30	60	20	80	40	~~90~~	~~10~~	
Next[i]	-1	5	6	1	2	-1	3	-1	-1	-1

Del 30

Head = 4, Size = 5



	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]	L[8]	L[9]
List[i]	~~50~~	70	~~30~~	60	20	80	40	~~90~~	~~10~~	
Next[i]	-1	5	-1	1	6	-1	3	-1	-1	-1

Linked List 鏈表

完成以下程式 1-10

```
#define MAX 10
char list[MAX][10];
int next[MAX];
int head, x, y,
listSize;
```

```
void ListInit(){
        int i;
        head=-1;
        for(i=0;i<MAX;i++){
                list[i]=_(1)_          // empty cell
                next[i]=-1;            // end of list
        }
        listSize=0;
}
```

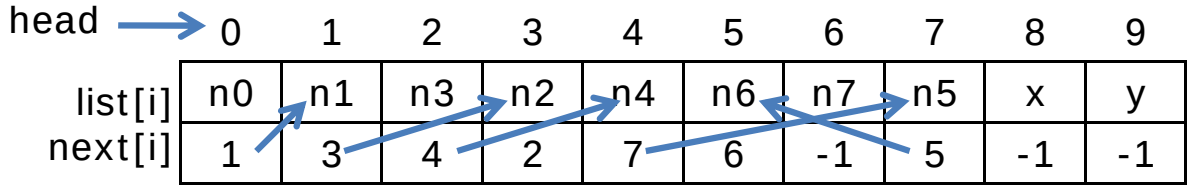
```
int ListEmpty(){
        return_(2)_;
}
```

```
int ListFull(){
        return_(3)_;
}
```

```
void displayListInfo(){
        gotoxy(50,02); printf("List Head = %2d", head);
        gotoxy(50,03); printf("List Size = %2d", listSize);
}
```

→

```
List Head = 0
List Size  = 8
```




```
void AddItem(char item[]){    // 新增
        int n, prev, curr;
        if(ListFull())
           printf("\n\nSorry - List Full!\n");
        else{
                n = nextAvailableCell();
                ___list[n]___item___          // (5)
                prev = curr = head;                  // find the insertion point
                while(curr!=-1 && strcmp(item,list[curr])___0){
                        // (6)
                        prev = curr;
                        curr = next[curr];
                }
                if(curr==head){                      // add to the front
                        head = n;
                        next[n] = (ListEmpty()?-1:curr);
                }else{
                        ___                  // (7)
                }
                listSize++;
        }
}
```

```
int nextAvailableCell(){      // 找空位
        int n=0;
        if(___) return -1;  // (4)
        while(___) ___;
        return n;
}
```

```
void ListAdd(){
        char item[10];
        printf("<< Add items to the List >>\n");
        y = whereY();

        do{
                gotoxy(1,y); printf("   => item to add : ");
                fflush(stdin); gets(item);
                y = whereY();
                if(strlen(item)>0) AddItem(item);
                displayListInfo();
        }while(strlen(item)>0 && !ListFull());
}
```

```
void listAll(){             // not in order
        int n;
        printf("\n<< Items currently in List are >>\n");
        for(n=0;n<MAX;n++)
                printf("<%2d> %-10s %d\n", n, list[n], next[n]);
        system("pause");
}
```



```
void DelItem(char item[ ]){             // 刪除 delete 1 item
        int n, prev, curr;
        if(ListEmpty())
                printf("\n\nSorry - List Empty!\n");
        else{
                prev = curr = head;             // locate the target item
                while(curr!=-1 && (n=strcmp(item,list[curr]))___0){   // (8)
                        prev = ___
                        curr = ___
                }

                if(n==0){                       // target found
                        if(curr==head)          // target = first item
                                head = ___              // (9)
                        else
                                next[___] = ___
                        printf("\n\nItem %10s has been removed.\n",item);
                        strcpy(list[curr],"");
                        next[curr]=___
                        listSize___
                }else
                        printf("\n\nSorry! item %s not found!\n", item);
        }
}
```

```
void ListDelete(){
        char item[10];
        printf("<< Delete items from the List >>\n");
        y = whereY();

        do{
                gotoxy(1,y); printf("   => item to delete : ");
                fflush(stdin); gets(item);
                y = whereY();

                if(strlen(item)>0) DelItem(item);
                displayListInfo();
        }while(strlen(item)>0 && !ListEmpty());
}
```



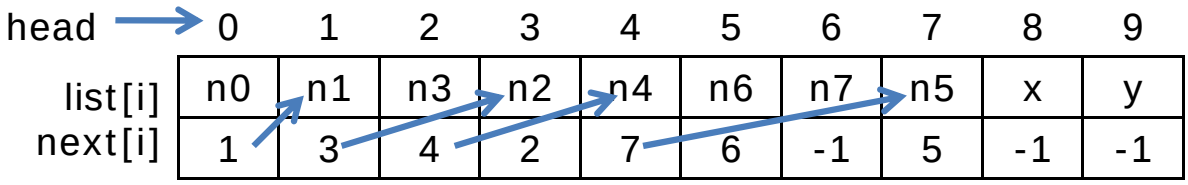
```
void displayList(){                 // ordered list
        int n;
        if(___)
                printf("\n\nSorry - List is empty !\n");          // (10)
        else{
                printf("\n\n<< Items currently in List are >>\n");

                n=head;
                while(___){
                        printf("<%2d> %-10s\n", n, list[n]);
                        n = ___
                }
        }
        system("pause");
}
```

```
void displayMenu(){
        printf(" <0> Initialize List  \n");
        printf(" <A> List Add         \n");
        printf(" <D> List Remove      \n");
        printf(" <L> Display List     \n");
        printf(" <T> Test Data        \n");
        printf(" <Q> Quit             \n");
}
```

```
void testdata(){
        strcpy(list[0],"n0");     next[0]=1;
        strcpy(list[1],"n1");     next[1]=3;
        strcpy(list[2],"n3");     next[2]=4;
        strcpy(list[3],"n2");     next[3]=2;
        strcpy(list[4],"n4");     next[4]=7;

        strcpy(list[5],"n6");     next[5]=6;
        strcpy(list[6],"n7");     next[6]=-1;
        strcpy(list[7],"n5");     next[7]=5;
        strcpy(list[8],"  ");     next[8]=-1;
        strcpy(list[9],"  ");     next[9]=-1;
        listSize=8;
        head=0;
        listAll();
}
```



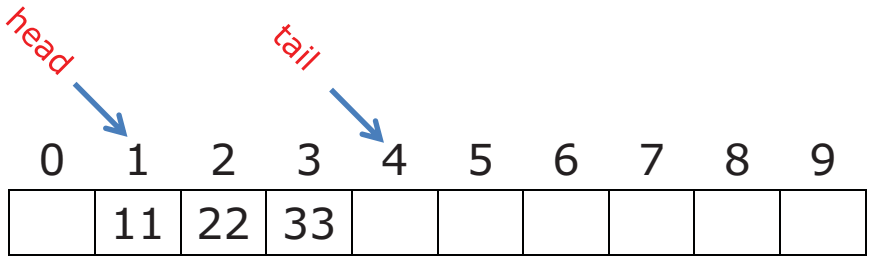

```
main(){
        char option; int valid;
        ListInit();

        do{
                clrscr();          displayMenu();
                printf("Which option <0-3,Q> ?  ");
                getxy(&x,&y);  displayListInfo();

                gotoxy(x,y);     option=getche();
                option=toupper(option); printf("\n\n");
                switch(option){
                        case'0': ListInit();       break;
                        case'A': ListAdd();        break;
                        case'D': ListDelete();     break;
                        case'L': displayList();    break;
                        case'X': listAll();        break;
                        case'T': testdata();       break;
                }
        }while(option!='Q');
}
```

Queue 隊列

```
int Queue[10];
int head, tail;
```

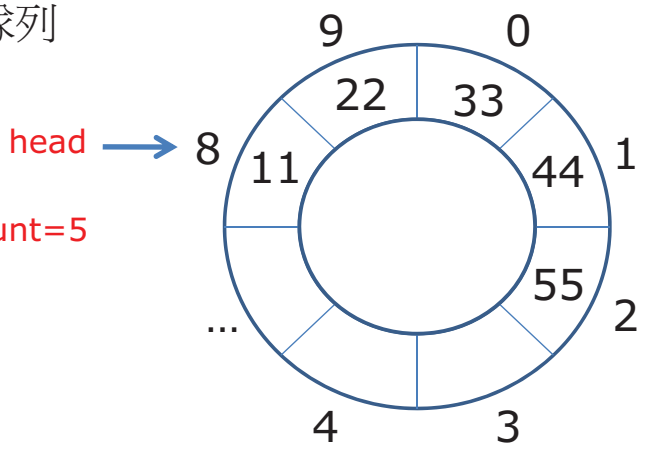



```
void listQueue(){
        int i;
        for(i=head;i<tail;i++)
                printf("%i\t",Queue[i]);
}
```


Circular Queue 循環隊列

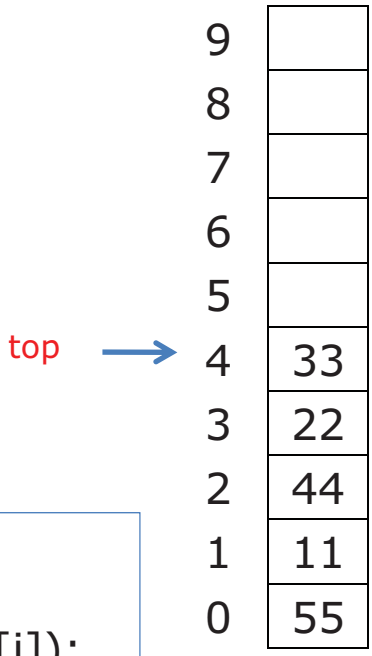
```
int Queue[10];
int head, count;
```

count=5




```
void listCQueue()
        int i;
        for(i=0;i<count;i++){
                n = (i+head)%10;
                printf("%i\t",Queue[n]);
        }
}
```

Stack 堆疊

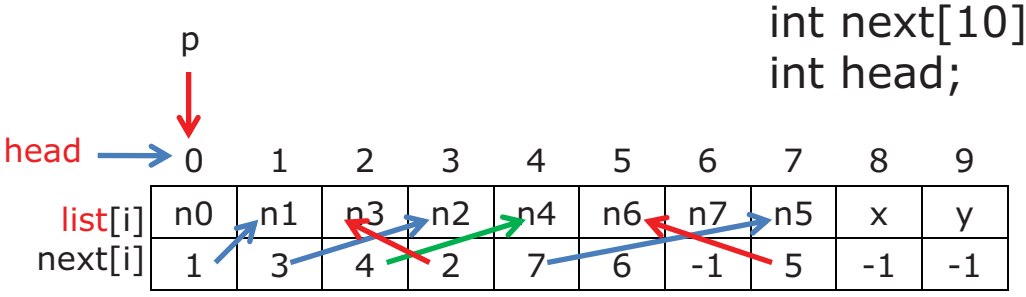



```
int Stack[10];
int top;
```

```
void displayStack()
        for(i=top; i>=0; i--)
                printf("%i\n",Stack[i]);
}
```


Linked List 鏈表

```
int list[10];
int next[10];
int head;
```




```
void displayLList()
        int p = head;
        while(p>-1){
                printf("%i\n",list[p]);
                p = next[p];
        }
}
```

資料結構 Data Structure

優點
1. 陣列 (Array)
   沒有額外變量 (例head,next,...)
   運作/操作較簡單
2. 隊列Queue (FIFO)
   容易更新
   加入enqueue: 隊尾tail
   移除dequeue: 隊首head
3. 堆疊Stack (LIFO)
   容易操作
   加入Push移除Pop皆在頂部stack top進行
4. 鏈表Linked List
   加新或移除資料時，
      只需改動少量資料，仍能保持順序
   動態結構Dynamic
      (程式執行期間，可隨意增減元素)

缺點
1. 陣列 (Array)
   更新資料時，若要保持順序，需要移動較大量資料
   靜態結構Static (宣告時已決定了陣列元素多少，
   不能在程式執行期間增減元素)
2. 隊列Queue (FIFO)
   需要額外變量head / tail
3. 堆疊Stack (LIFO)
   需要額外變量top
4. 鏈表Linked List
   需要額外變量head, next
   運作/操作較複雜(搜尋、更新)

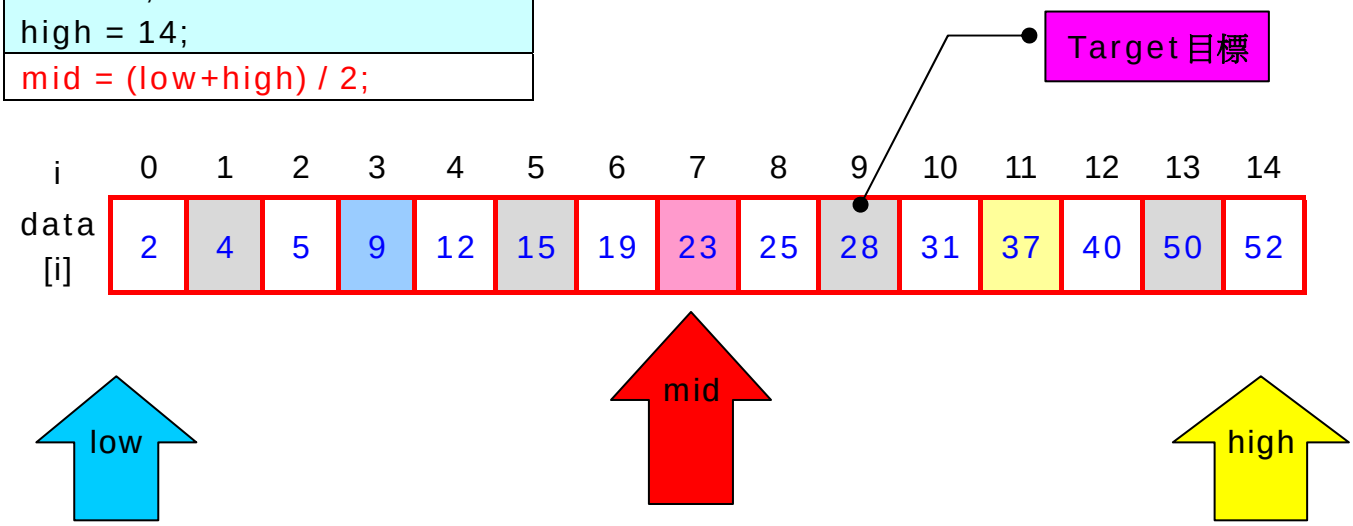
動態配置Dynamic Memory Allocation

```
int *ptr = malloc(sizeof(int));
printf("空間位置   ：%X\n", ptr);
printf("空間儲存值：%d\n", *ptr);
*ptr = 200;
printf("空間位置   ：%X\n", ptr);
printf("空間儲存值：%d\n", *ptr);
free(ptr);
```

Binary Search: 對分檢索法

```
int data[15];
int low, mid, high, target;
```

```
low = 0;
high = 14;
mid = (low+high) / 2;
```



target = 28							
pass	low	mid	high	data[mid]	target	?	data[mid]
1	1	8	15	23	target		data[mid]
2					target		data[mid]
3					target		data[mid]
4							

target = 4							
pass	low	high	mid	data[mid]	target	?	data[mid]
1	1	8	15	23	target		data[mid]
2					target		data[mid]
3					target		data[mid]
4							

target = 11							
pass	low	high	mid	data[mid]	target	?	data[mid]
1	1	8	15	23	target		data[mid]
2					target		data[mid]
3					target		data[mid]
4					target		data[mid]

target (11) NOT found!

Binary Search: 對分檢索法

```
int data[15];

low = 0;
high = 14;
pos = -1;

while (low<=high && pos==-1) {
    mid = (low+high) / 2;
    if ( target == data[mid] ) pos = mid;
    else if ( target < data[mid] ) high = mid-1;
        else low = mid+1;
}
```

Sequential Search: 順序檢索

```
int data[15], pos = -1;

for ( n=0; n<15; n++ )
    if ( target==data[n] ) break;
if ( target==data[n] ) pos = n;      // if ( n<15 ) pos = n;
```

Bubble Sort: 冒泡排序法

```
char x[100][20];
```

```
for ( pass=1; pass<99; pass++)
    for ( i=0; i<99-pass; i++ )
        if (x[i] > x[i+1]) {
            t = x[i];
            x[i] = x[i+1];
            x[i+1] = t;
        }
```

```
pass = 1;
do{
    ok = 1;
    for ( i=0; i<99-pass; i++ )
        if (x[i] > x[i+1]) {
            t = x[i];
            x[i] = x[i+1];
            x[i+1] = t;
            ok = 0;
        }
    pass = pass+1;
while (ok == 0);
```